CSC110 Fall 2021 Assignment 2: Logic, Constraints, and Nested Data

TODO: FILL IN YOUR NAME HERE

October 10, 2021

Part 1: Predicate Logic

1. 1. Statement 1 is saying that "given a set of numbers, one number will always be greater than all the numbers in the set including itself". Statement 2 is saying "given a set of numbers, one number will always be less than all the numbers in the set including itself". It's obvious that both of these statements are false since a number can never be greater than or less than itself. Therefore, there does not exist a set D_1 that can make one statement true and the other false.
   2. Same reasoning as [1], it's impossible for there to be a set D_2 that can make both statements true.
   3. Same reasoning as [1], any set can make both statement 1 and statement 2 false. Take $\{1\}$ as D_3. For statement 1, 1 cannot be greater than all the numbers in the set since 1 cannot be greater than itself. For statement 2, 1 cannot be less than all the numbers in the set since 1 cannot be less than itself. Therefore, there exists a set D_3 that makes both statements false.

2. 1. P(x): "$x < 5$," where $x \in S$.
   2. Q(x): "$x < 6$," where $x \in S$.

   Since the set includes all integers from 0 to 9 inclusive, it's impossible for all of them to be less than 5 and less than 6. Take 8, for example. However, we can assert that if any value in the set is less than 5, then it will also be less than 6(since 6 is greater than 5). Therefore the implication that if an element from S is less than 5, then it will be less than 6 will hold true.

3. Complete this part in the provided `a2_part1.py` starter file. Do **not** include your solution in this file.

4. Complete this part in the provided `a2_part1.py` starter file. Do **not** include your solution in this file.

Part 2: Conditional Execution

Complete this part in the provided `a2_part2.py` starter file. Do **not** include your solution in this file.

Part 3: Generating a Timetable

1. Complete this part in the provided `a2_part3.py` starter file. Do **not** include your solution in this file.

2. (a) *IMPORTANT DEFINITIONS/NOTATION* (don't change this text!)
   We define the following sets:
   - C: the set of all possible courses
   - S: the set of all possible sections
   - M: the set of all possible meeting times
   - SC: the set of all possible schedules

   We also define the following notation for expressions involving the elements of these sets:
   - The first three (courses/sections/meeting times) are represented as tuples (as described in the assignment handout), and you can use the indexing operation on these values. For example, you could translate "every section term is in $\{'F', 'S', 'Y'\}$" into predicate logic as the statement:

     $$\forall s \in S,\ s[1] \in \{'F', 'S', 'Y'\}$$

   - The start and end times of a meeting time can be compared chronologically using the standard $<$, $\leq$, $>$, and $\geq$ operators.
   - For a section $s \in S$, $s[2]$ represents a tuple of meeting times. You may use standard set operations and quantifiers for these tuples (pretend they are sets). For example, we can say:
     - $\forall s \in S,\ s[2] \subseteq M$
     - $\forall s \in S,\ \forall m \in s[2],\ m[1] < m[2]$
   - Finally, for a schedule $sc \in SC$, you can use the notation $sc.sections$ to refer to a set of all sections in that schedule. You can use quantifiers with that set of schedules as well, e.g. $\forall s \in sc.sections,\ ...$

   Predicate for meeting times conflicting:

   $$MeetingTimesConflict(m_1, m_2) : m_1[0] = m_2[0] \wedge ((m_1[1] \leq m_2[1] < m_1[2]) \vee (m_2[1] \leq m_1[1] < m_2[2])) \text{ where } m_1, m_2 \in M$$

   Predicate for sections conflicting:

   $$SectionsConflict(s_1, s_2) : \exists m_1 \in s_1[2]\ s.t.\ \exists m_2 \in s_2[2]\ s.t.\ MeetingTimesConflict(m_1,\ m_2) \wedge (s_1[1] = s_2[1]\ \vee\ s_1 =' Y'\ \vee\ s_2 =' Y') \text{ where } s_1, s_2 \in S$$

   Predicate for valid schedule:

   $$IsValidSchedule(sc) : \forall s_1,\ s_2 \in sc,\ s_1 = s_2\ \vee\ \neg SectionsConflict(sc[s_1],\ sc[s_2]) \text{ where } sc \in SC$$

   (b) Complete this part in the provided `a2_part3.py` starter file. Do **not** include your solution in this file.

3. (a) You may use all notation from question 2(a). Note that a course $c \in C$ is a tuple, and $c[2]$ is a set of sections, and so can be quantified over: $\forall s \in c[2], ...$

   Predicate for section-schedule compatibility:

   $$IsCompatibleSection(sc, s) : \forall s_0 \in sc,\ \neg SectionsConflict(sc[s_0],\ s) \text{ where } sc \in SC, s \in S$$

   Predicate for course-schedule compatibility:

   $$IsCompatibleCourse(sc, c) : \exists s_1 \in c[2]\ s.t\ \forall s_2 \in sc,\ \neg SectionsConflict(s_1,\ sc[s_2]) \text{ where } sc \in SC, c \in C$$

   (b) Complete this part in the provided `a2_part3.py` starter file. Do **not** include your solution in this file.

Part 4: Processing Raw Data

Complete this part in the provided `a2_part4.py` starter file. Do **not** include your solution in this file.